Module 01 – Fundamentals

**Table of Content**

[AWS Regions & AZs 1](#_Toc198379641)

[AWS EC2 1](#_Toc198379642)

[EC2 Service 1](#_Toc198379643)

[EC2 Launch Types 1](#_Toc198379644)

[Amazon Machine Image (AMIs) 1](#_Toc198379645)

[Placement Groups 1](#_Toc198379646)

[Elastic Network Interfaces (ENIs) 1](#_Toc198379647)

[EC2 Hibernate 1](#_Toc198379648)

[AWS ELB 1](#_Toc198379649)

[ELB Service 1](#_Toc198379650)

[Types of Load Balancers 1](#_Toc198379651)

[ELB Features 1](#_Toc198379652)

[AWS ASG 1](#_Toc198379653)

[ASC Service 1](#_Toc198379654)

[Scaling Features 1](#_Toc198379655)

[AWS EBS 1](#_Toc198379656)

[EBS Service 1](#_Toc198379657)

[EBS Snapshots 1](#_Toc198379658)

[Instance Store 2](#_Toc198379659)

[Elastic File System 3](#_Toc198379660)

[Relational Database Service 4](#_Toc198379661)

[Amazon Aurora 5](#_Toc198379662)

[ElasticCache 6](#_Toc198379663)

[Amazon S3 7](#_Toc198379664)

[Route 53 8](#_Toc198379665)

# AWS Regions & AZs

### Regions

Think of AWS Regions as neighborhoods of data centers spread across the globe. Each Region is a collection of physical locations where AWS builds and maintains its infrastructure. These are labeled with names like us-east-1 (Virginia, USA) or eu-west-1 (Ireland).

Most AWS services are region-scoped, which means the data and resources you create in one Region stay in that Region unless you intentionally replicate them elsewhere. So, if you launch a server in us-west-2, it won’t automatically show up in us-east-1—they’re completely independent environments.

### Availability Zones (AZs)

Now, let’s zoom into a Region. Each AWS Region is divided into smaller parts called Availability Zones, or AZs. These are like separate buildings within the same neighborhood. A Region typically has 3 AZs, but it can have anywhere from 2 to 6.

Each AZ consists of one or more physically distinct data centers, each with its own power supply, networking, and connectivity, built to be highly reliable.

Even though AZs within a Region are geographically separated—to protect against issues like power outages or natural disasters—they’re also closely connected through high-speed, low-latency networking. This means you can design systems that are both resilient and fast, using multiple AZs within the same Region.

# AWS EC2

## EC2 Service

AWS EC2 (Elastic Compute Cloud) is one of the core services in the AWS ecosystem. It allows you to run virtual machines in the cloud, giving you flexible computing power without needing to buy physical hardware.

Here are the main things you can do with EC2:

* **Rent Virtual Machines (EC2 Instances)**: You can launch servers—called instances—on demand. These act like traditional computers, but they run in the cloud and can be started, stopped, resized, or terminated whenever you need.
* **Store Data on Virtual Drives (EBS)**: EC2 works with Elastic Block Store (EBS) to provide storage. Think of EBS volumes like external hard drives that attach to your EC2 instances, allowing you to store data persistently—even after the instance shuts down.
* **Distribute Traffic Across Multiple Machines (ELB)**: With Elastic Load Balancing (ELB), you can automatically spread incoming traffic across several EC2 instances. This helps your applications stay responsive and available, even during high demand.
* **Automatically Scale with Demand (Auto Scaling Group - ASG)**: An Auto Scaling Group (ASG) monitors your instances and automatically adds or removes them based on traffic or other conditions. This way, your system can handle traffic spikes and save costs during low activity.

### Security Groups

Security Groups are a core part of network security in AWS. You can think of them as virtual firewalls that protect your EC2 instances. They control what kind of traffic is allowed in or out of your cloud servers.

In simpler terms: they help answer the question, *"Who can talk to my EC2 instance, and on which port?"*

Security Groups work by setting rules that define how network traffic flows to and from your EC2 instances. Here's what they do:

* **Control Access to Specific Ports:** For example, port 22 for SSH, port 80 for HTTP, or port 443 for HTTPS.
* **Allow or Deny IP Ranges:** You can specify who gets access using IPv4 or IPv6 address ranges.
* **Manage Inbound and Outbound Traffic:** Decide what traffic can come in (inbound) or go out (outbound) of your EC2 instance.

A few important notes about security groups:

* **Reusable Across Instances:** One security group can be attached to multiple EC2 instances, so you don’t need to create a new one for every server.
* **Scoped to Region and VPC:** Security Groups are limited to a specific Region and Virtual Private Cloud (VPC). You can’t share them across regions.
* **External to EC2:** Security Groups operate outside the EC2 instance. If a request is blocked, the EC2 instance won’t even know it happened—like a bouncer turning someone away at the door.

Here are some tips when working with security groups:

* **Create a Separate SG for SSH:** It’s a good practice to isolate SSH access in its own security group. This makes it easier to manage and audit access.
* **Diagnosing Access Issues:**
* If your application request times out, it’s likely that traffic is being blocked by a Security Group rule.
* If you get a “connection refused” error, it usually means the traffic made it through, but the application itself isn't responding correctly.
* **Default Rules:**
* By default, all inbound traffic is denied.
* All outbound traffic is allowed—unless you explicitly block it.
* **Referencing Other Security Groups:** You don’t have to use IP addresses. A security group can allow traffic from another security group directly, which is especially helpful when working with multiple tiers (like web and database servers).

### Elastic IPs

When you launch an EC2 instance, AWS assigns it a public IP address so it can communicate with the internet. But here’s the catch: if you stop and restart that instance, it will likely get a new public IP. This can be a problem if you need the instance to always be reachable at the same IP address.

That’s where Elastic IPs come in.

An Elastic IP is a static public IP address that you can own and control within your AWS account. Once you allocate an Elastic IP, it’s yours until you release (delete) it. You can associate it with any EC2 instance in your account, and even move it between instances if one fails—helping you quickly recover from outages.

Think of it as your permanent street address in the cloud, even if you change the house (the EC2 instance) it points to.

A few important things to know:

* AWS gives you a soft limit of 5 Elastic IPs per region by default.
* Elastic IPs are not free if they’re allocated but not actively used (i.e., not associated with a running instance). AWS encourages efficient use of resources.

In most cases, it's better to avoid Elastic IPs unless you have a specific need for a static IP. Here's why:

* They can indicate a rigid or outdated architecture, especially in modern, scalable cloud environments.
* A better practice is to use a random public IP for your EC2 instance and point a DNS name (like a domain name) to it. This way, even if the IP changes, you can just update the DNS record—keeping your setup more flexible and cloud-native.

### EC2 User Data

When you launch an EC2 instance, you often want it to be ready to go—without having to log in and set things up manually. That’s where EC2 User Data comes in.

User Data is a way to automate the initial setup of your instance by running a script at launch.

You can use a User Data script to perform common setup tasks, such as:

* Installing system updates
* Installing necessary software (like web servers or databases)
* Downloading files from the internet
* Running custom startup commands—basically, anything you’d normally do right after logging in

These scripts run automatically when the instance starts for the first time, and they’re executed with root (administrator) privileges, so they can make system-level changes.

Key Points to Remember

* **Runs Only Once:** The User Data script runs only during the first boot of the instance (unless you explicitly configure it to run every time).
* **Great for Automation:** It helps you save time, avoid manual errors, and create consistent setups for multiple instances.
* **Ideal for Launching Pre-Configured Servers:** For example, you can spin up a web server that’s fully installed and ready to serve traffic in just a few minutes.

## EC2 Launch Types

When launching EC2 instances, AWS offers several pricing and deployment options—each suited for different use cases. Here's an overview of the main types:

### On-Demand Instances

* **Pay-per-use model**: You’re billed by the second after the first minute.
* No upfront payment required.
* **Best for:** Short-term, unpredictable workloads or development and testing environments.
* **Pros:** Flexibility and no commitment.
* **Cons:** Highest cost compared to other options.

### Reserved Instances (RIs)

Reserved Instances offer significant savings in exchange for committing to a specific instance type over a long term.

**Types of Reserved Instances:**

* **Standard Reserved Instances**
  + Save up to 75% compared to On-Demand.
  + Fixed commitment for 1 or 3 years.
  + Recommended for: Steady, long-term workloads (e.g., databases, production apps).
* **Convertible Reserved Instances**
  + Save up to 54%.
  + Allows changing instance types, OS, or tenancy.
  + Best for: Long-term workloads that may change over time.
* **Scheduled Reserved Instances** *(less common)*
  + Run during specific time windows (e.g., 9 AM–5 PM weekdays).
  + Useful when workloads only run at scheduled times.

### Spot Instances

* Get up to 90% discount compared to On-Demand.
* AWS uses unused capacity, but instances can be interrupted at any time.
* **Best for:** Fault-tolerant and stateless workloads (e.g., batch jobs, data analysis).
* **Not recommended for:** Critical applications or databases.
* **Max Price:** You define the maximum price you're willing to pay.
* **Termination Notice:** AWS gives a 2-minute warning if your instance is being reclaimed.
* **Spot Block:** Lock a Spot instance for 1–6 hours without interruptions (rarely reclaimed).
* **Spot Request:** Specify pricing, instance type, number of instances, and request type:
  + **One-time:** Fills once, then disappears.
  + **Persistent:** Keeps trying to maintain your desired capacity.
* **Canceling Spot Requests:** Cancel the request before terminating the instance, or it may relaunch.

### Dedicated Instances

* EC2 instances that run on hardware dedicated to your AWS account.
* Other AWS customers won’t share the same physical host, but instances from your account can.
* Per-instance billing, but no control over where instances are placed.

### Dedicated Hosts

* You get an entire physical server.
* Allows full visibility into the underlying hardware, including sockets and cores.
* Required for software licenses tied to physical hardware (e.g., Oracle, Microsoft).
* 3-year reservation required.
* Best for: Compliance-heavy environments and software with strict licensing rules.

### Spot Fleet

A Spot Fleet is a group of Spot Instances (with optional On-Demand instances) that AWS uses to meet your desired capacity at the lowest possible cost.

* AWS selects from multiple launch pools (combinations of instance types, OS, and Availability Zones).
* You define:
  + Capacity goal
  + Pricing constraints
  + Instance configurations

**Allocation Strategies:**

* **Lowest Price**: Choose instances from the cheapest pool.
* **Diversified**: Spread instances across multiple pools for redundancy.
* **Capacity Optimized**: Focus on pools with more available capacity to reduce interruptions.

### EC2 Instance Families (By Use Case)

|  |  |  |
| --- | --- | --- |
| **Type** | **Best For** | **Example Use Case** |
| R | Memory-intensive | In-memory databases (e.g., Redis) |
| C | Compute-optimized | High-performance compute, CPU-bound tasks |
| M | General purpose | Balanced CPU and memory – ideal for web apps |
| I | Storage-optimized | High local storage throughput – databases |
| G | GPU-based | Machine learning, video rendering |
| T2/T3 | Burstable performance | Spiky workloads like development environments |

### Burstable Instances (T2/T3)

Burstable instances offer baseline performance with the ability to temporarily burst to higher CPU power when needed.

* Use CPU credits to burst.
* Earn credits during idle time; spend credits during bursts.
* If credits run out, performance drops to the baseline.
* Monitor usage with CloudWatch.

### T2/T3 Unlimited

* Allows unlimited bursting without being throttled, even if you run out of credits.
* You’ll be charged extra for CPU usage beyond earned credits.
* Ideal for unpredictable workloads where performance consistency is important.

## Amazon Machine Image (AMIs)

When you launch a virtual server (an EC2 instance) on AWS, you need something to base it on—this is where **Amazon Machine Images**, or **AMIs**, come in.

AWS provides a wide variety of **base AMIs**—these are pre-configured images with different operating systems and basic settings. Think of them like starter templates.

You can customize these images when the instance boots up using **EC2 User Data**. This allows you to run scripts and install software automatically at launch.

But sometimes, you need even more control and efficiency. That’s when you create your **own custom AMI**.

### Custom AMI

Creating a custom AMI gives you several advantages:

* **Pre-installed software**: You can include all the packages and tools your application needs ahead of time.
* **Faster boot time**: Since everything is already set up in the image, the instance doesn’t have to run long setup scripts every time it starts.
* **Standardized configuration**: You can include monitoring tools, enterprise software, or security agents so that every instance is configured exactly the same way.
* **Better security**: You have full control over what's inside the image and can ensure it meets your organization's security standards.
* **Simplified maintenance**: Updates and patches can be baked into the image, making maintenance easier.
* **Built-in Active Directory integration**: If your systems use Active Directory, you can pre-configure that in the AMI.

On a last note, AMIs are **region-specific**. That means if you create an AMI in one AWS region, it won’t automatically be available in others. You’ll need to copy it to other regions if needed.

## Placement Groups

When you're running applications on AWS, sometimes it's not just about **what** you're running but also **where** and **how** your EC2 instances are physically placed within AWS's infrastructure. That’s where **Placement Groups** come into play.

Placement Groups let you influence the placement of a group of EC2 instances to meet specific performance or availability requirements. When you create a placement group, you choose one of three placement strategies:

### Cluster Placement Group

This strategy places your instances close together within a single Availability Zone (AZ), on the same rack or very nearby. The goal is low latency and high network performance. The only downside is that this leads to a single point of failure so if the racks fail, all instances will go down together.

### Spread Placement Group

In a spread placement group, your instances are spread out across multiple physical machines, and potentially multiple Availability Zones. The only downside is that you are limited to placing 7 instances per AZ. It is ideal for high available systems where no two instances should fail at once. It is also useful if you require instance isolation.

### Partition Placement Group

This strategy divides your instances into logical partitions, where each partition is placed on distinct racks and infrastructure within a single AZ. It is most often used in Distributed systems like HDFS, HBase, Cassandra, or Kafka where spreading data across isolated groups of nodes is critical.

## Elastic Network Interfaces (ENIs)

An Elastic Network Interface (ENI) is like a virtual network card for your EC2 instances—except it's not tied to one machine and can be managed independently. It’s a key component of networking within a Virtual Private Cloud (VPC) in AWS.

### ENI Features

Each ENI comes with several networking features:

* A primary private IPv4 address
* Optionally, one or more secondary private IPv4 addresses
* One Elastic IP address (IPv4) for each private IP
* One public IPv4 address

This makes ENIs flexible and powerful for managing network connections in the cloud.

### ENI & EC2 Instances

* ENIs are not tied to EC2 instances by default. You can create them independently.
* You can attach an ENI to an instance at any time, or even move it between instances—this is especially useful for failover scenarios, where you want to quickly recover from a server issue.
* However, ENIs are specific to an Availability Zone. You can't move one across zones.
* You can also attach security groups directly to ENIs, giving you more control over traffic flow and security.

### ENI Attachment & Interfaces

Every EC2 instance automatically gets a primary ENI, usually assigned to eth0. This primary interface:

* Cannot be detached
* Is essential for the instance’s network connectivity

You can also attach secondary ENIs. These appear as eth1, eth2, and so on. They are useful if your instance needs to connect to multiple networks or handle more network traffic.

### Quick Use Cases

* **Failover setups**: Move an ENI to a backup instance to quickly recover from a failure.
* **Multiple IPs**: Run multiple services on the same instance with different IPs.
* **Network monitoring or security appliances**: Add additional ENIs for traffic mirroring or segmentation.

## EC2 Hibernate

When working with EC2 instances, you usually have two options when you're done using them:

* Stop the instance: The virtual machine powers down, but the data on your Elastic Block Store (EBS) volumes remains intact.
* Terminate the instance: The instance is permanently deleted, and the root EBS volume (where the operating system is installed) is also destroyed—unless you've configured it to be preserved.

Now, let’s talk about a third option: **Hibernate**.

Hibernate is like putting your laptop to sleep instead of shutting it down. It saves the contents of the instance’s RAM (memory) to disk, so when you restart the instance, it picks up right where it left off.

### Hibernation Process

* The RAM state is saved to a file on the root EBS volume.
* When you start the instance again, it skips the usual OS boot process and resumes from the saved memory state.
* This means your applications don’t need to restart, caches stay warm, and everything feels much faster.

### Use Cases

* **Long-running processes**: Avoid restarting apps or reloading data after every stop.
* **Development/testing environments**: Save time by resuming from where you left off.
* **Cost control**: Stop the instance and avoid charges, then resume work quickly when needed.

# AWS ELB

## ELB Service

When designing cloud-based applications, two key goals often come up: scalability and high availability. Elastic Load Balancers (ELBs) play a major role in achieving both.

### Scalability

Scalability means that a system can handle more load as demand increases by adjusting its resources accordingly.

There are two main types of scalabilities:

* + **Vertical Scalability (Scale *up* or *down*)**: This means upgrading the size or power of a single instance.
  + **Horizontal Scalability (Scale *out* or *in*)**: This involves adding more instances to distribute the workload.

### Availability

High Availability (HA) means your application is designed to keep running even if one data center (Availability Zone, or AZ) goes down.

There are two types of HA setups:

* **Passive**: A standby system takes over when the primary one fails.
* **Active**: All systems run simultaneously and share the load.

To be highly available, your application should run across multiple AZs—not just one.

### Elastic Load Balancers

A load balancer is a server that distributes incoming internet traffic to multiple backend servers (usually EC2 instances). Think of it as the traffic cop at the entrance to your application.

A few features are ELBs include:

* **Distribute traffic evenly** to prevent any one server from being overloaded.
* **Expose a single DNS name** (like myapp.com) while managing multiple backend instances.
* **Automatically detect unhealthy instances** and reroute traffic to healthy ones.
* **Terminate SSL** (HTTPS) connections, offloading that work from your app servers.
* **Enable sticky sessions** (using cookies) to keep users connected to the same backend instance.
* **Improve availability** by spanning across multiple AZs.
* **Separate public and private traffic**, improving security and architecture.

## Types of Load Balancers

AWS offers four types of load balancers, each designed for different use cases and levels of network traffic. Think of them as different tools in your cloud toolkit—choose the one that best fits the job.

* 1. **Classic Load Balancer (CLB)** – *Old Generation*
     + Supports: HTTP, HTTPS, and TCP
     + It was the original load balancer in AWS and works well for basic use cases.
     + However, it lacks advanced features like routing rules or container support.
  2. **Application Load Balancer (ALB)** – *New Generation*
     + Supports: HTTP, HTTPS, and WebSockets.
     + Ideal for web applications.
     + Offers advanced routing features based on path and host.
     + Supports microservices and containers
  3. **Network Load Balancer (NLB)** – *New Generation*
     + Supports: TCP, TLS (encrypted TCP), and UDP
     + Designed for high-performance, low-latency use cases.
     + Can handle millions of requests per second while maintaining ultra-low latency.
     + Ideal for real-time applications, gaming, IoT, or financial systems.
  4. **Gateway Load Balancer (GWLB)** – *New Generation*
     + Integrates with third-party network appliances like firewalls or intrusion detection systems.
     + Works at the network layer (Layer 3) and routes traffic to appliance fleets inside your VPC.
     + Useful for organizations needing deep packet inspection, traffic filtering, or advanced security processing.

## ELB Features

### Cross-Zone Load Balancing

By default, a load balancer sends traffic only to the targets (like EC2 instances) within the same AZ as the load balancer node. But with cross-zone load balancing enabled, the load balancer can distribute traffic evenly across all available AZs, regardless of which AZ the request came through.

### SSL/TLS Certifications

When your users connect to your application, especially over the internet, security is essential. That's where SSL/TLS certificates come into play—they encrypt the connection between the user (client) and your load balancer.

* SSL (Secure Sockets Layer) was the original standard for encrypting data sent over the internet.
* TLS (Transport Layer Security) is the newer, more secure version—but people often still say “SSL” out of habit.
* Encryption using SSL/TLS is called encryption in transit or in-flight encryption.

### Server Name Indication (SNI)

Traditionally, a web server could only serve one SSL certificate per IP address. This was limiting—especially for hosting multiple secure websites on a single server.

SNI is a modern extension to the TLS protocol that fixes this limitation.

* During the initial SSL handshake, the client tells the server which hostname it's trying to reach.
* This lets the server select the correct SSL/TLS certificate from multiple options, even when using the same IP address.

Think of it like a customer entering a bakery and saying, “I’m here to pick up the chocolate cake,” so the baker knows which cake to hand over—even if there are many.

### ELB Connection Draining (a.k.a. Deregistration Delay)

When you’re removing an instance from a load balancer—maybe for maintenance or scaling down—you don’t want it to suddenly drop active user connections. That’s where connection draining comes in.

It:

* **Stops new requests** from being sent to the instance.
* **Allows in-flight requests** (ongoing work) to complete before the instance is fully removed from service.

This graceful exit helps prevent data loss, errors, or interrupted user sessions.

# AWS ASG

## ASC Service

Imagine your application is like a busy restaurant. Some nights, it's packed—you need more staff. Other times, it's quiet—so fewer people are needed. Auto Scaling Groups (ASGs) help your cloud-based applications handle similar patterns of demand by automatically adjusting the number of EC2 instances running behind the scenes.

With ASG enabled, you can:

* **Scale out**: Add more EC2 instances when traffic increases.
* **Scale in**: Remove instances when demand drops.
* **Maintain limits**: Always keep at least a minimum number—and never more than a maximum—of instances running.
* **Stay connected**: Automatically register new instances with a load balancer so traffic is distributed smoothly.

### ASG w/ CloudWatch Alarms

ASGs can react to **CloudWatch alarms**, which monitor specific metrics like:

* Average CPU usage
* Network traffic
* Disk I/O

When these metrics cross thresholds (e.g., CPU > 70%), the ASG can trigger:

* **Scale-out policies** to add more instances
* **Scale-in policies** to remove instances

## Scaling Features

### Scaling Policies

There are a few types of scaling policies you can use:

* 1. Target Tracking Scaling
     + Keeps a metric (e.g., CPU) at a set target.
     + Easiest to use and most automated.
  2. Simple or Step Scaling
     + Manual thresholds and actions.
     + Example: If CPU > 70%, add 2 instances. If < 30%, remove 1.
  3. Scheduled Actions
     + Pre-plan scaling based on known usage patterns
     + Example: Scale up every Friday at 5 PM.

### Scaling Cooldowns

After a scaling action, it takes time for the effect to show up. To prevent rapid, unnecessary scaling, ASGs use cooldown periods:

* Default: 300 seconds (5 minutes)
* You can override this with scaling-specific cooldowns
* Useful for fine-tuning cost and performance, especially when scaling in

# AWS EBS

## **EBS Service**

When you launch an EC2 instance, it runs on a virtual machine in the cloud. But what happens to your data when the instance shuts down—especially if it’s terminated unexpectedly?

That’s where EBS (Elastic Block Store) comes in. Think of EBS as a cloud-based hard drive that you can attach to your EC2 instances to store data persistently—even after the instance is stopped or terminated.

### **EBS Workings**

* **EBS is a network drive**:
  + It connects to your EC2 instance over the network, so some latency is expected compared to local storage.
  + You can detach it from one instance and attach it to another, if both are in the same Availability Zone (AZ).
* **AZ-bound**:
  + EBS volumes are locked to a single AZ. To move them elsewhere, you must take a snapshot first.
* **Provisioned capacity**:
  + You choose the size (in GB) and performance (IOPS) when creating a volume.
  + Billing is based on provisioned size, not actual usage.

## EBS Snapshots

Snapshots are like saving a backup copy of your drive:

* **Incremental**: Only the changes since the last snapshot are saved.
* **Stored in S3**: But you won’t see them directly in your S3 bucket.
* **No need to detach**, but it’s safer to snapshot when the app isn’t under heavy load.
* **Snapshots can be used to create AMIs** or to move volumes across AZs or regions.
* **Restored volumes need pre-warming** (e.g., using fio or dd commands) for best performance.
* Use **Amazon Data Lifecycle Manager** to automate snapshot creation and deletion.

### Migrating EBS Volumes Across AZs or Regions

Because EBS volumes are tied to an AZ, you can’t just “move” them. Instead, follow these steps:

1. Create a snapshot of the volume.
2. *(Optional)* Copy the snapshot to another region.
3. Create a new volume from the snapshot in your desired AZ.

### EBS Encryption

Encryption ensures your data is secure—both **at rest** and **in transit**.

* When you create an encrypted volume, everything is encrypted:
  + Data on disk
  + Data moving between the instance and EBS
  + Snapshots
  + Volumes created from snapshots
* EBS uses AES-256 and is integrated with AWS Key Management Service (KMS).
* Encryption is transparent—you don’t need to manage it yourself.
* You can encrypt an unencrypted volume by:
  + Creating a snapshot of the volume
  + Copying the snapshot and enabling encryption during the copy
  + Creating a new volume from that snapshot

## Instance Store

Some EC2 instances come with instance store instead of EBS. Here’s how they compare:

An instance store is:

* Physically attached to the host machine (not over the network)
* Blazing fast I/O performance
* Great for temporary data like caches, buffers, or scratch files
* Data survives reboots, but data is lost when the instance stops or terminates.

# **Elastic File System**

…

# **Relational Database Service**

…

# **Amazon Aurora**

…

# **ElasticCache**

…

# **Amazon S3**

…

# **Route 53**

…